Software Testing Strategies

Peter Lo

Overview

- Integrates software test case design techniques into a well-planned series of steps that result in the successful construction of software.
- A testing strategy must always incorporate test planning, test case design, text execution, and the resultant data collection and evaluation

Generic Characteristics of Software Testing Strategies

- Testing beings at the module level and works toward the integration of the entire system
- Different testing techniques are appropriate at different points in time
- Testing is conducted by the software developer and an independent test group (ITG)
- Debugging must be accommodated in any testing strategy

Verification and Validation

- **Verification** - Set of activities that ensure that software correctly implements a specific function
  - i.e. Are we building the project right?
- **Validation** - Set of activities that ensure that the software that has been built is traceable to customer requirements
  - i.e. Are we building the right product?

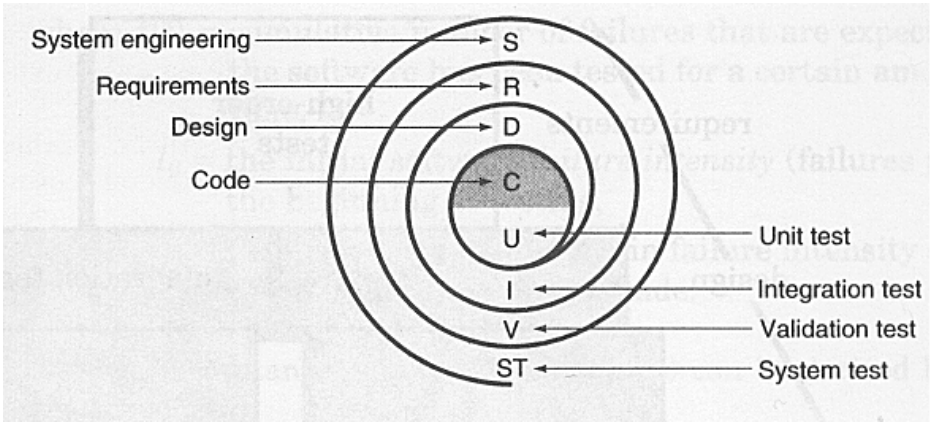
Misconceptions in Software Testing

- Software developer should not do any testing at all
- The software should be “throw” to the testers who will test it mercilessly
- Testers get involved with the project only when the testing steps are about to begin.

Misconceptions Refutation

- Software developer test the individual units of the program, ensuring that each performs the function for which it was designed
- Independent Test Group remove the inherent problems associated with letting the building test the thing that has been built
- The software developer should correct errors that are uncovered once testing starts.

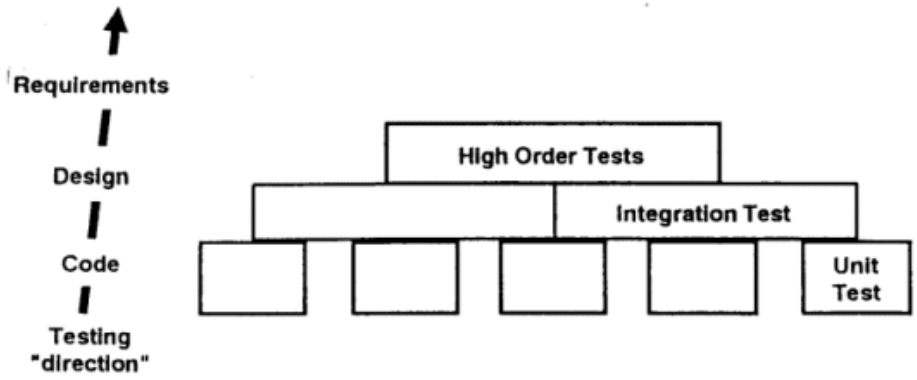
Software Testing Strategy



Software Testing Strategy

- A strategy for software testing moves outward along the spiral.
  - Unit Testing: concentrates on each unit of the software as implemented in the source code.
  - Integration Testing: focus on the design and the construction of the software architecture.
  - Validation Testing: requirements established as part of software requirement analysis are validated against the software that has been constructed.
  - System Testing: the software and other system elements are tested as a whole.

Testing Direction



Software Testing Direction

- Unit Tests
  - Focuses on each module and makes heavy use of white box testing
- Integration Tests
  - Focuses on the design and construction of software architecture; black box testing is most prevalent with limited white box testing.
- High-order Tests
  - Conduct validation and system tests. Makes use of black box testing exclusively.

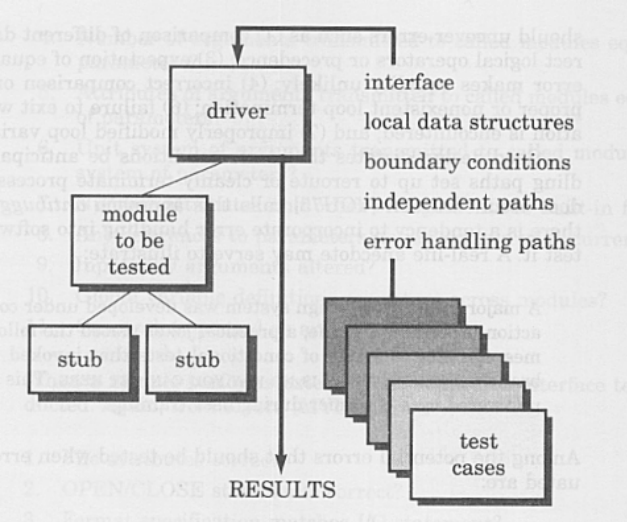
Unit Testing

- Unit testing focuses verification effort on the smallest unit (module) of software design
- Important control paths are tested to uncover errors within the boundary of the module by using the detail design description as a guide
- The unit test is always white box-oriented

Unit Testing Procedures

- Module is not a stand-alone program, driver & stub software must be developed for each unit test.
- A driver is a program that accepts test case data, passes such data to the module, and prints the relevant results.
- Stubs serve to replace modules that are subordinate the module to be tested. A stub or "dummy subprogram" uses the subordinate module's interface, may do nominal data manipulation, prints verification of entry, and returns.
- Drivers and stubs also represent overhead

Unit Testing Environment



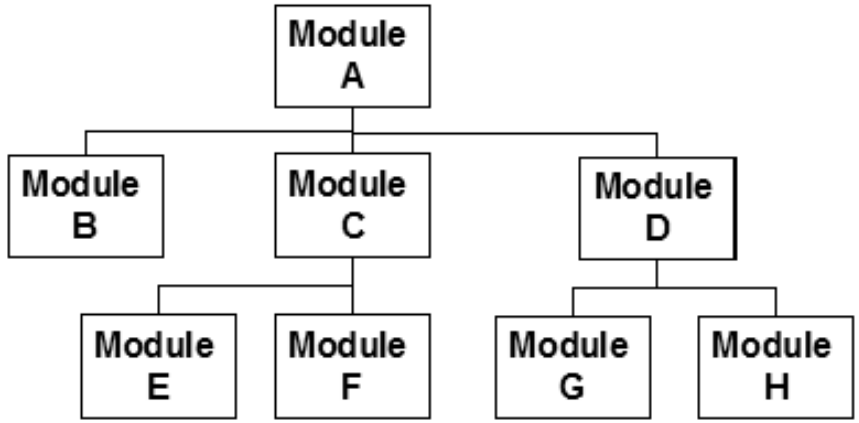
Integration Testing

- Integration Testing is a technique for constructing the program structure while at the same time conducting tests to uncover tests to uncover errors associated with interfacing
- Objective is combining unit-tested modules and build a program structure that has been dictated by design.

Top-Down Integration Testing

- The main control module is used as a test driver and stubs are substituted for all modules directly subordinate to the main control module
- Subordinate stubs are replaced one at a time with actual modules
- Tests are conducted as each module is integrated
- On the completion of each set of tests, another stub is replaced with the real module
- Regression testing may be conducted to ensure that new errors have not been introduced

Top-Down Testing Example



Top-Down Testing Example

- For the above program structure, the following test cases may be derived if top-down integration is conducted:
  - Test case 1: Modules A and B are integrated
  - Test case 2: Modules A, B and C are integrated
  - Test case 3: Modules A., B, C and D are integrated (etc.)

Problem of Top-Down Testing

- There is a major problem in top-down integration:
  - Inadequate testing at upper levels when data flows at low levels in the hierarchy are required

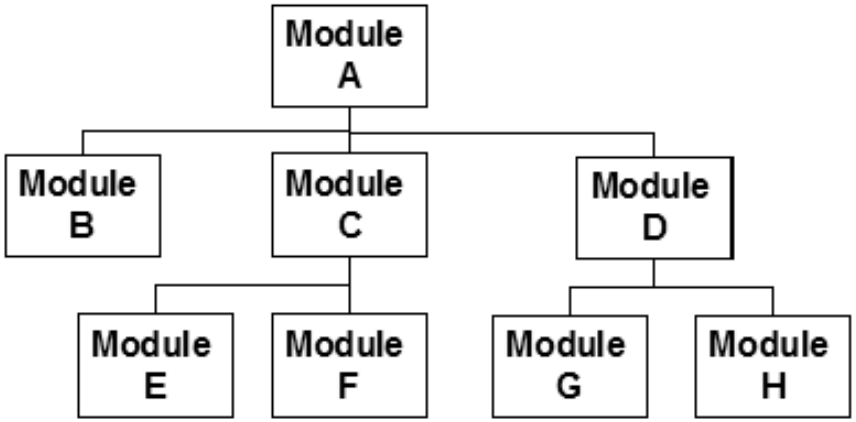
Solutions of Top-Down Testing

- Delay many test until stubs are replaced with actual modules; but this can lead to difficulties in determining the cause of errors and tends to violate the highly constrained nature of the top-down approach
- Develop stubs that perform limited functions that simulate the actual module; but this can lead to significant overhead
- Perform bottom-up integration

Bottom-Up Testing

- Low-level modules are combined into clusters that perform a specific software sub-function
- A driver is written to coordinate test case input and output
- The cluster is tested
- Drivers are removed and clusters are combined moving upward in the program structure

Bottom-Up Testing Example



Bottom-Up Testing Example

- Test case 1: Modules E and F are integrated
- Test case 2: Modules E, F and G are integrated
- Test case 3: Modules E., F, G and H are integrated
- Test case 4: Modules E., F, G, H and C are integrated (etc.)
- Drivers are used all round.

Validating Testing

- Ensuring software functions can be reasonably expected by the customer.
- Achieve through a series of black tests that demonstrate conformity with requirements.
- A test plan outlines the classes of tests to be conducted, and a test procedure defines specific test cases that will be used in an attempt to uncover errors in conformity with requirements.
- A series of acceptance tests are conducted with the end users

Validating Testing

- Alpha testing
  - Is conducted at the developer's site by a customer
  - The developer would supervise
  - Is conducted in a controlled environment
- Beta testing
  - Is conducted at one or more customer sites by the end user of the software
  - The developer is generally not present
  - Is conducted in a "live" environment

System Testing

- Since software is only one component of a larger system. A series of system integration and validation tests are conducted once software is incorporated with other system elements
- System testing is a series of different tests whose primary purpose is to fully exercise the computer-based system.
- Although each system test has a different purpose, all work to verify that all system elements have been properly integrated and perform allocated functions.

Recovery Testing

- A system test that forces software to fail in a variety of ways and verifies that recovery is properly performed
- If recovery is automatic, re-initialization, check-pointing mechanisms, data recovery, and restart are each evaluated for correctness
- If recovery is manual, the mean time to repair is evaluated to determine whether it is within acceptable limits.

Security Testing

- Security testing attempts to verify that protection mechanisms built into a system will in fact protect it from improper penetration.
- Particularly important to a computer-based system that manages sensitive information or is capable of causing actions that can improperly harm individuals when targeted.

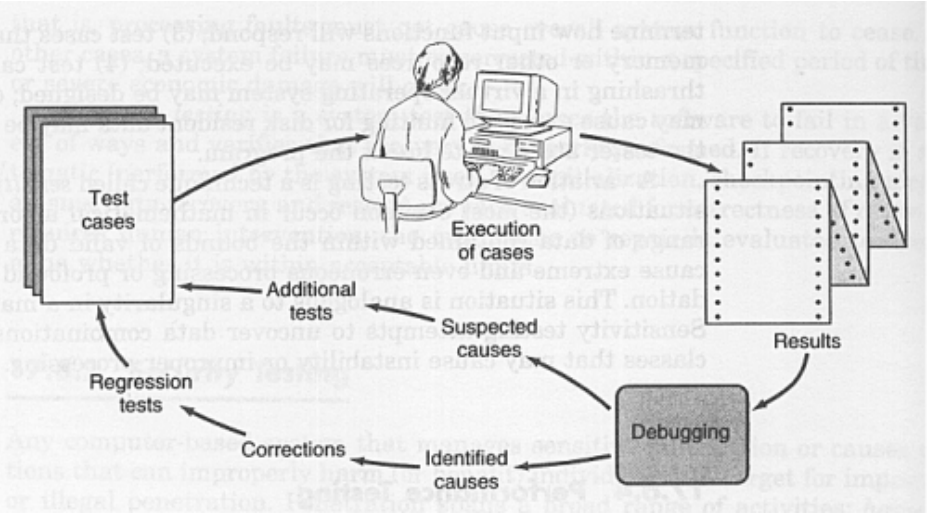
Stress Testing

- Stress Testing is designed to confront programs with abnormal situations where unusual quantity frequency, or volume of resources are demanded
- A variation is called sensitivity testing;
  - Attempts to uncover data combinations within valid input classes that may cause instability or improper processing

Performance Testing

- Test the run-time performance of software within the context of an integrated system.
- Extra instrumentation can monitor execution intervals, log events as they occur, and sample machine states on a regular basis
- Use of instrumentation can uncover situations that lead to degradation and possible system failure

Debugging



Debugging

- Debugging will be the process that results in the removal of the error after the execution of a test case.
- There are two possible outcomes of the debugging process:
  - The cause of the error will be found, corrected, and removed;
  - The cause of the error will not be found.

Characteristics of Bugs

- The symptom and the cause may be geographically remote
- The symptom may disappear when another error is corrected
- The symptom may actually be caused by non-errors
- The symptom may be caused by a human error that is not easily traced
- It may be difficult to accurately reproduce input conditions
- The symptom may be intermittent. This is particularly common in embedded systems that couple hardware and software inextricably
- The symptom may be due to causes that are distributed across a number of tasks running on different processors

Debugging Approaches

- Brute force
  - The program is loaded with run-time traces, and WRITE statements, and hopefully some information will be produced that will indicated a clue to the cause of an error.
- Backtracking
  - Starting from where the symptom has been uncovered, backtrack manually until the site of the cause is found.
- Cause Elimination
  - Data related to the error occurrence is organized to isolate potential causes. A "cause hypothesis" is devised and the above data are used to prove or disapprove the hypothesis.

Additional Debugging Considerations

- Once a bug is found, it must be corrected.
- Important to keep in mind still:
  - Is the cause of the bug reproduced in another part of the program?
  - What “next bug” might be introduced by the fix that I’m about to make?
  - What could we have done to prevent this bug in the first place?

Debugging Tools

- Debugging compilers
- Dynamic debugging aides ("tracers")
- Automatic test case generators
- Memory dumps